Metro Bundler micro-surgery

by Yury Pliashkou

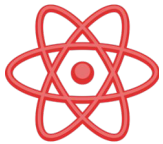
"Metro is a JavaScript bundler for React Native."

Fast

Metro aims for sub-second reload cycles, fast startup and quick bundling speeds.

Scalable

Works with thousands of modules in a single application.



Integrated

Supports every React Native project out of the box.

Features

Build-in server

The server has the ability to serve assets, bundles and source maps for those bundles.

Assets

In order to request an asset, you can freely use the `require` method as if it was another JS file. The server will treat this specific `require` calls and make them return the path to that file. When an asset is requested (an asset is recognized by its extension, which has to be on the `assetExts` array) it is generally served as-is.

However, the server is also able to serve specific assets depending on the platform and on the requested size (in the case of images). The way you specify the platform is via the dotted suffix (e.g. **`.ios`**) and the resolution via the at suffix (e.g. **`@2x`**). This is transparently handled for you when using `require`.

Bundle

Any JS file can be used as the root for a bundle request. The file will be looked in the `projectRoot`. All files that are required by the root will be recursively included. In order to request a bundle, just change the extension from `.js` to `.bundle`. Options for building the bundle are passed as query parameters (all optional).

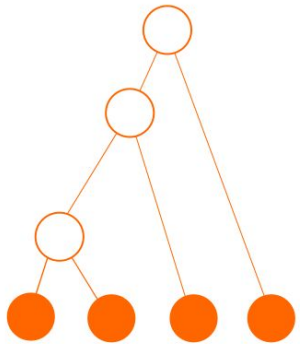
- **`dev:`** build the bundle in development mode or not. Maps 1:1 to the `dev` setting of the bundles. Pass `true` or `false` as strings into the URL.
- **`platform:`** platform requesting the bundle. Can be `ios` or `android`. Maps 1:1 to the `platform` setting of the bundles.
- **`minify:`** whether code should be minified or not. Maps 1:1 to the `minify` setting of the bundles. Pass `true` or `false` as strings into the URL.
- **`excludeSource:`** whether sources should be included in the source map or not. Pass `true` or `false` as strings into the URL.

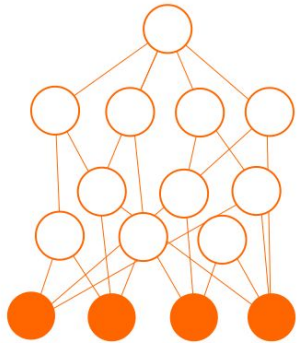
For instance, requesting **`http://localhost:8081/foo/bar/baz.bundle?dev=true&platform=ios`** will create a bundle out of `foo/bar/baz.js` for iOS in development mode.

Bundle creation steps

Resolution

Metro needs to build a graph of all the modules that are required from the entry point. To find which file is required from another file Metro uses a resolver. In reality this stage happens in parallel with the transformation stage.



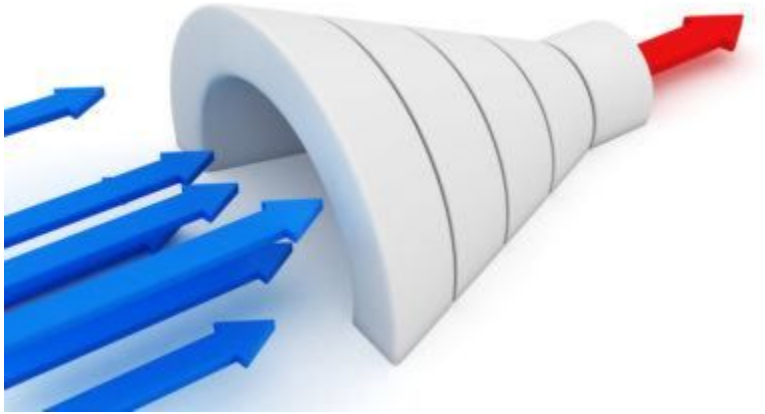


Transformation

All modules go through a transformer. A transformer is responsible for converting (transpiling) a module to a format that is understandable by the target platform (eg. React Native). Transformation of modules happens in parallel based on the amount of cores that you have. Usually Babel is used.

Serialization

As soon as all the modules have been transformed they will be serialized. A serializer combines the modules to generate one or multiple bundles. A bundle is literally a bundle of modules combined into a single JavaScript file



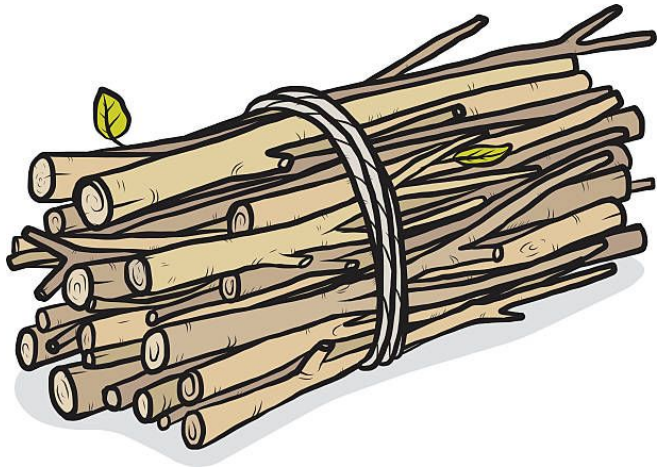
Bundling

When bundling, each of the modules gets assigned a numeric id, meaning no dynamic requires are supported. Requires are changed by its numeric version, and modules are stored in different possible formats. Three different formats of bundling are supported:

- Plain bundle
- File RAM bundle
- Indexed RAM bundle

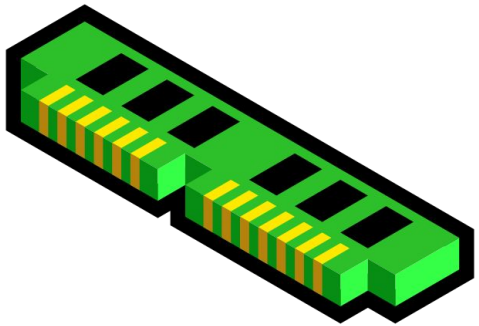
Plain bundle

This is the standard bundling format. In this format, all files are wrapped with a function call, then added to the global file. This is useful for environments that expect a JS only bundle (e.g. a browser). Just requiring the entry point with the `.bundle` extension should trigger a build of it.



File RAM bundle

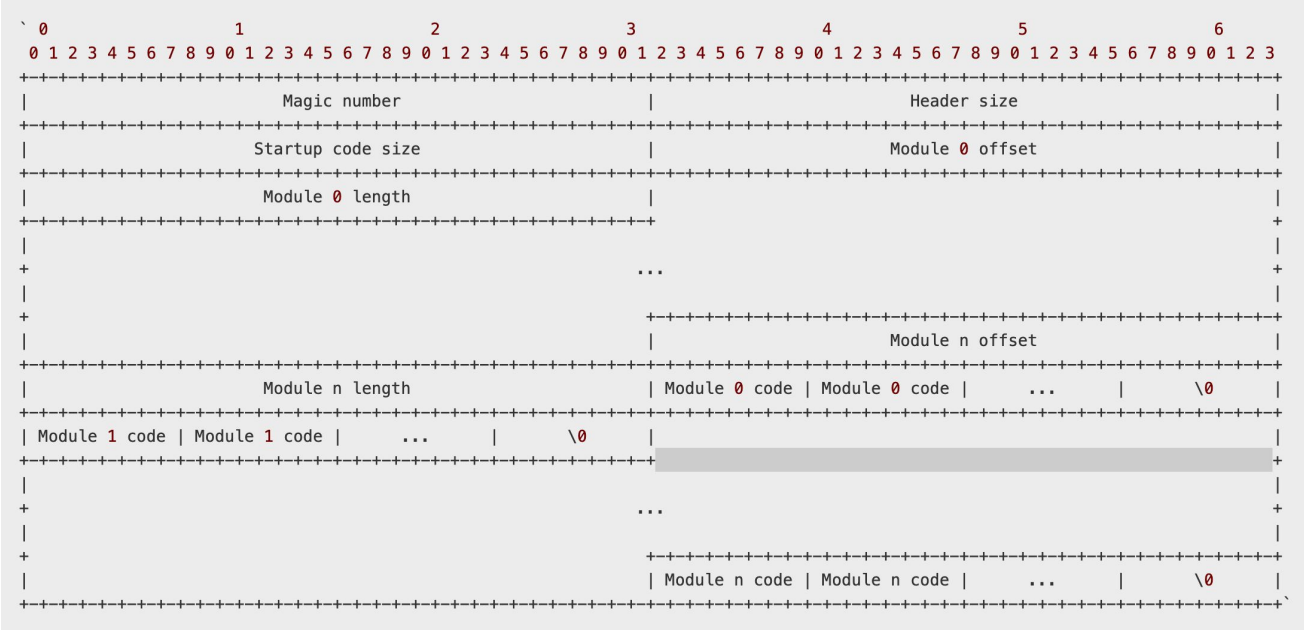
Each module is stored as a file, with the name `js-modules/${id}.js`, plus an extra file called `UNBUNDLE` is created, which its only content is the magic number, `0xFB0BD1E5`. Note that the `UNBUNDLE` file is created at the root. This bundling is usually used by Android, since package contents are zipped, and access to a zipped file is much faster. If the indexed format was used instead, all the bundled should be unzipped at once to get the code for the corresponding module.



Indexed RAM bundle

This format composes the bundle as a binary file, which format has the following parts (all numbers are expressed in Little Endian):

- A magic number: a `uint32` must be located at the beginning of the file, with the value `0xFB0BD1E5`. This is used to verify the file.
- An offset table: the table is a sequence of `uint32` pairs, with a header
  - For the header, two `uint32`s can be found: the length of the table, and the length of the startup code.
  - For the pairs, they represent the offset in the file and the length of code module, in bytes.
- Each of the modules, finished by a null byte (`\0`).



This structure is optimal for an environment that is able to load all code in memory at once:

- By using the offset table, one can load any module in constant time, where the code for module `x` is located at `file[(x + 3) * sizeof(uint32)]`. Since there is a null character (`\0`) separating all modules, usually length does not even need to be used, and the module can be loaded directly as an ASCIIZ string.
- Startup code is always found at `file[sizeof(uint32)]`.

It's time for coding!

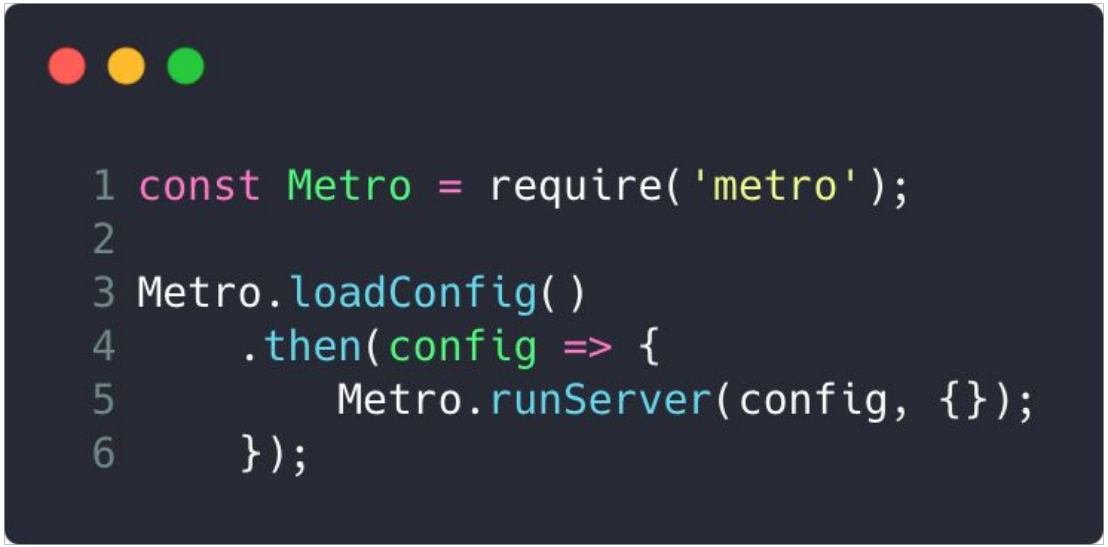

auto
cache
dist
node_modules
public
src
javascript.js
typescript.tsx
.babelrc
.gitignore
build.js
metro.config.js
package.json
server.js
yarn.lock
1 alert('JavaScript is ..!!');
1 {
2 "dependencies": {
3 "@babel/preset-env": "^7.6.3",
4 "@babel/preset-react": "^7.6.3",
5 "@types/react-dom": "^16.9.2",
6 "express": "^4.17.1",
7 "metro": "^0.56.0",
8 "metro-core": "^0.56.0",
9 "react": "^16.10.2",
10 "react-dom": "^16.10.2"
11 }
12 }

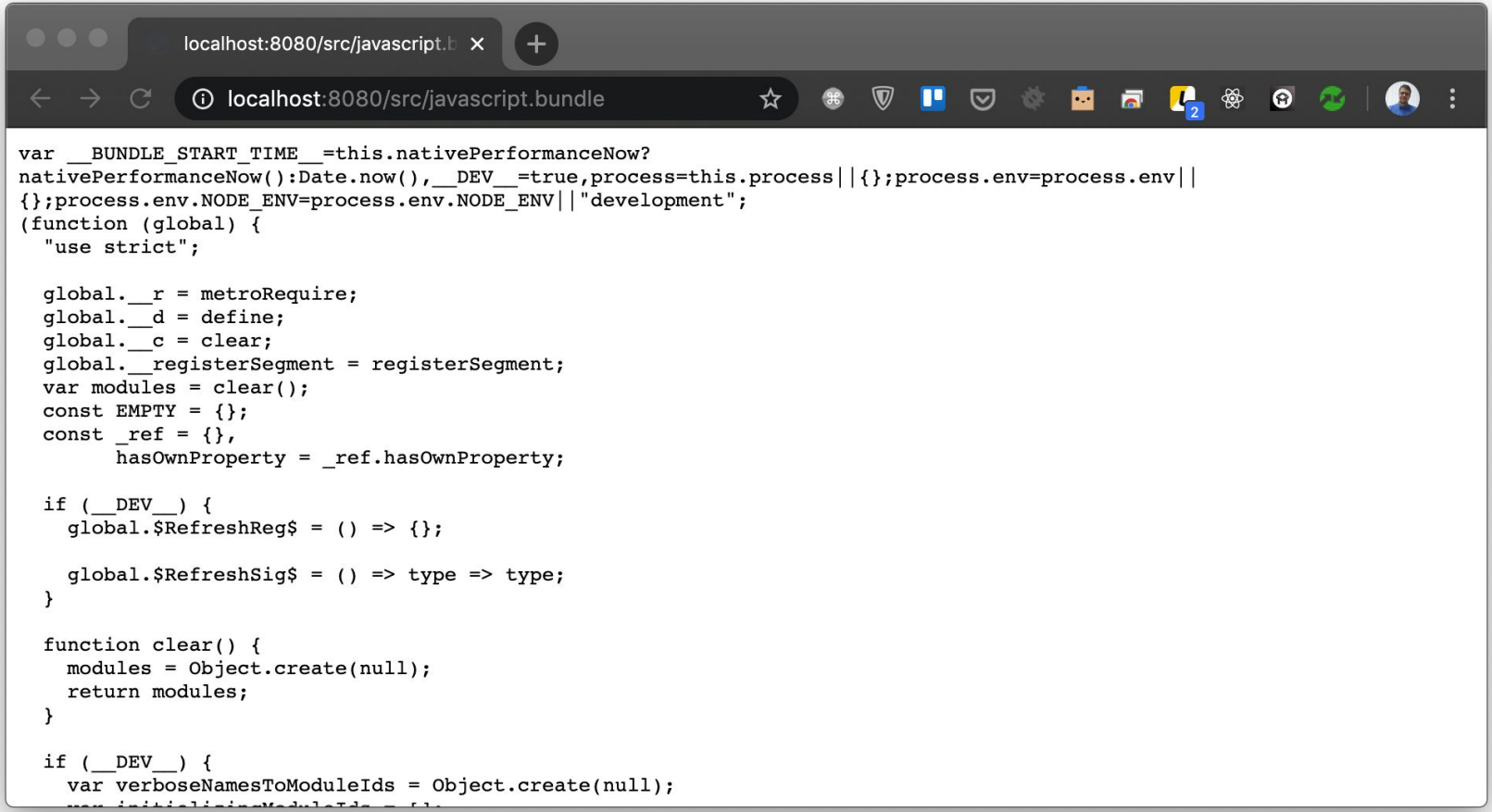

const Metro = require('metro')
Metro.loadConfig().then(config => {
Metro.runBuild(config, {
entry: './src/javascript.js',
out: './dist/javascript.js'
})
})

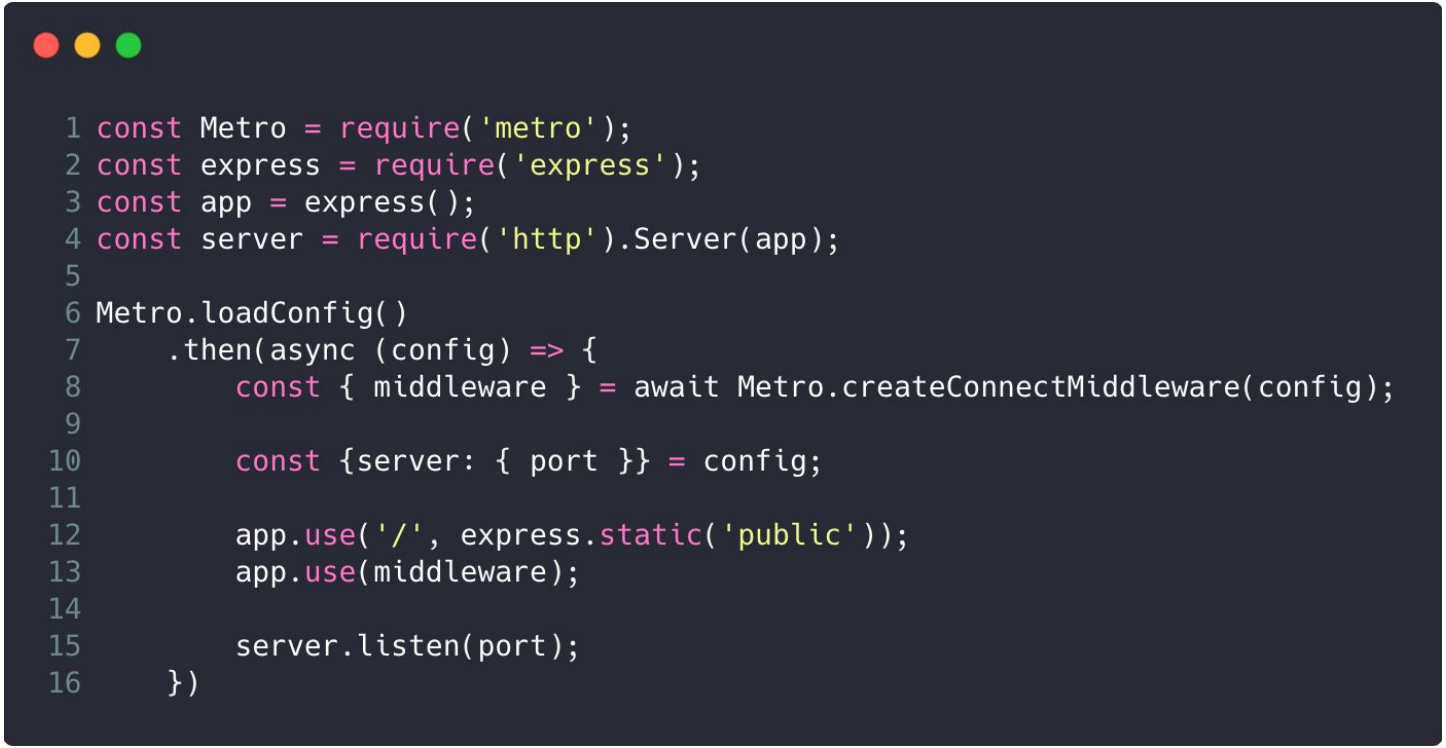
```
1 alert('JavaScript is ..!!');
```

30 bytes -> 2200 bytes = Profit!

```
1 var __BUNDLE_START_TIME__=this.nativePerformanceNow?
  nativePerformanceNow():Date.now(),__DEV__=false,process=this.process||{};process.env=process.env||
  {};process.env.NODE_ENV=process.env.NODE_ENV||"production";
2 !(function(t){"use strict";t.__r=i,t.__d=function(t,n,o){if(null!=e[n])return;const i=
  {dependencyMap:o,factory:t,hasError:!1,importedAll:r,importedDefault:r,isInitialized:!1,publicModule:{exports:
  {}}};e[n]=i},t.__c=o,t.__registerSegment=function(t,e){p[t]=e};var e=o();const r={},n={}.hasOwnProperty;function
  o(){return e=Object.create(null)}function i(t){const r=t,n=e[r];return n&&n.isInitialized?
  n.publicModule.exports:d(r,n)}function l(t){const n=t;if(e[n]&&e[n].importedDefault!==r)return
  e[n].importedDefault;const o=i(n),l=o&&o.__esModule?o.default:o;return e[n].importedDefault=l}function u(t){const
  o=t;if(e[o]&&e[o].importedAll!==r)return e[o].importedAll;const l=i(o);let u;if(l&&l.__esModule)u=l;else{if(u=
  {},l)for(const t in l)n.call(l,t)&&(u[t]=l[t]);u.default=l}return
  e[o].importedAll=u}i.importDefault=l,i.importAll=u;let c=!1;function d(e,r){if(!c&&t.ErrorUtils){let
  n;c=!0;try{n=m(e,r)}catch(e){t.ErrorUtils.reportFatalError(e)}return c=!1,n}return m(e,r)}const
  s=16,a=65535;function f(t){return{segmentId:t>>>s,localId:t&a}}i.unpackModuleId=f,i.packModuleId=function(t)
  {return(t.segmentId<<s)+t.localId};const p=[];function m(r,n){if(!n&&p.length>0){const
  t=f(r),o=t.segmentId,i=t.localId,l=p[o];null!=l&&(l(i),n=e[r])}const o=t.nativeRequire;if(!n&&o){const
  t=f(r),i=t.segmentId;o(t.localId,i),n=e[r]}if(!n)throw h(r);if(n.hasError)throw
  I(r,n.error);n.isInitialized=!0;const c=n,d=c.factory,s=c.dependencyMap;try{const e=n.publicModule;return
  e.id=r,d(t,i,l,u,e,e.exports,s),n.factory=void 0,n.dependencyMap=void 0,e.exports}catch(t){throw
  n.hasError=!0,n.error=t,n.isInitialized=!1,n.publicModule.exports=void 0,t}}function h(t){return Error('Requiring
  unknown module "'+t+'".')}function I(t,e){return Error('Requiring module "'+t+'", which threw an exception:
  '+e)}})('undefined'!=typeof globalThis?globalThis:'undefined'!=typeof global?global:'undefined'!=typeof window?
  window:this);
3 __d(function(g,r,i,a,m,e,d){"use strict";alert('JavaScript is ..!!')},0,[]);
4 __r(0);
```


const Metro = require('metro');
Metro.loadConfig()
.then(config => {
Metro.runServer(config, {});
});


localhost:8080/src/javascript.b
localhost:8080/src/javascript.bundle
var __BUNDLE_START_TIME__=this.nativePerformanceNow?
nativePerformanceNow():Date.now(),__DEV__=true,process=this.process||{};process.env=process.env||
{};process.env.NODE_ENV=process.env.NODE_ENV||"development";
(function (global) {
"use strict";
global.__r = metroRequire;
global.__d = define;
global.__c = clear;
global.__registerSegment = registerSegment;
var modules = clear();
const EMPTY = {};
const _ref = {},
hasOwnProperty = _ref.hasOwnProperty;
if (__DEV__) {
global.$RefreshReg$ = () => {};
global.$RefreshSig$ = () => type => type;
}
function clear() {
modules = Object.create(null);
return modules;
}
if (__DEV__) {
var verboseNamesToModuleIds = Object.create(null);

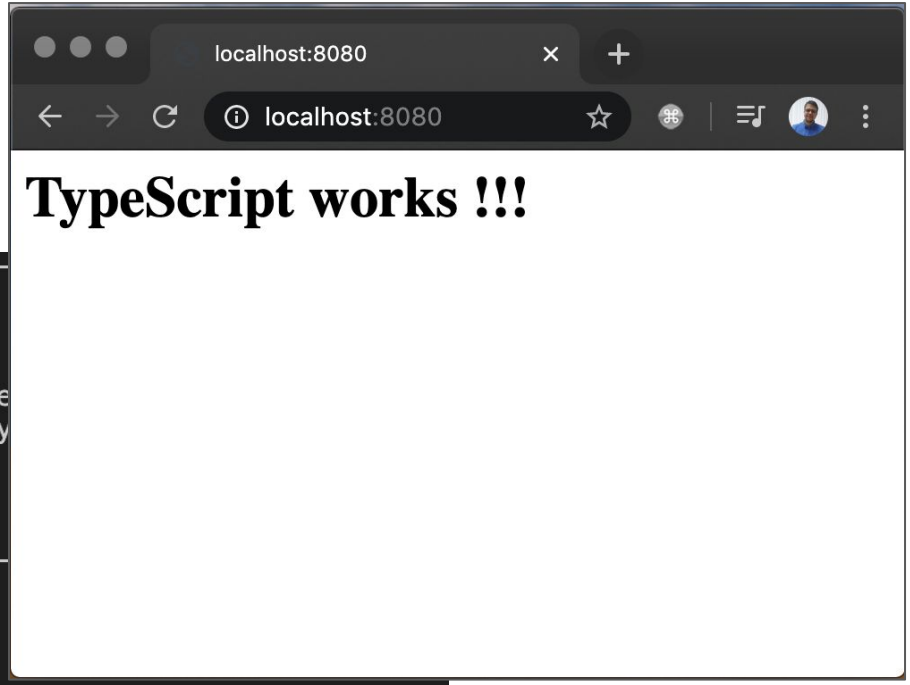


```
const Metro = require('metro');
const express = require('express');
const app = express();
const server = require('http').Server(app);

Metro.loadConfig()
    .then(async (config) => {
        const { middleware } = await Metro.createConnectMiddleware(config);

        const {server: { port }} = config;

        app.use('/', express.static('public'));
        app.use(middleware);

        server.listen(port);
    })
```

```tsx
import * as React from 'react';
import * as ReactDOM from 'react-dom';

function HelloWorld() {
    return (
        <h1>TypeScript works !!!</h1>
    );
}

ReactDOM.render(
    <HelloWorld />,
    document.getElementById('root')
);
```

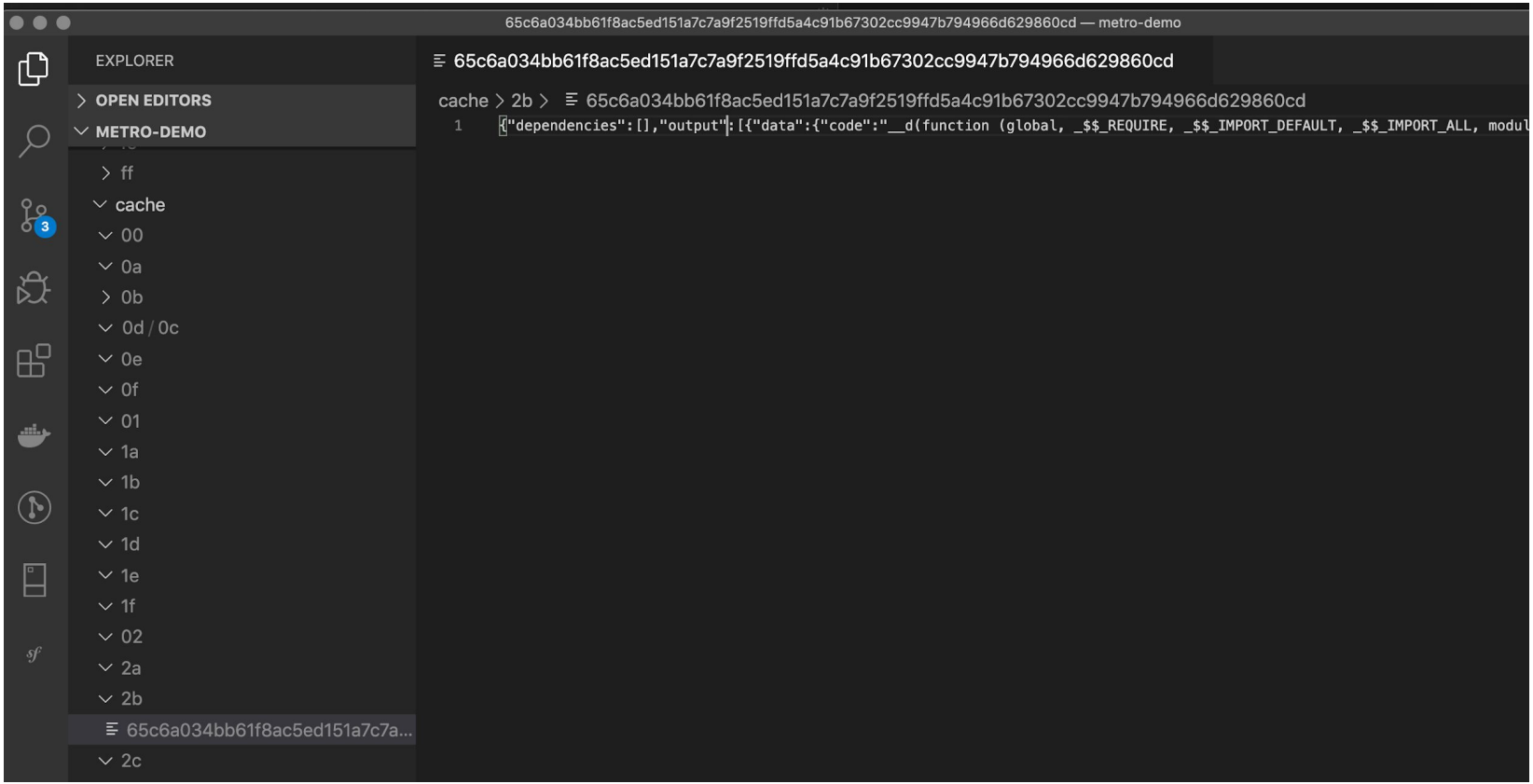
```html
<html>
    <body>
        <div id="root"></div>
        <script src="/src/typescript.bundle"></script>
    </body>
</html>
```

133Kb (React ~ 10kb + React-DOM ~ 120kb) = 40Kb gzipped




```
Running Metro Bundler on port 8080.

Keep Metro running while developing on any JS proje
close this tab and run your own Metro instance if y

https://github.com/facebook/react-native

Looking for JS files in
   /Users/lynx/workspace/metro-demo

Loading dependency graph, done.
BUNDLE  [dev] src/javascript.js  100.0% (1/1), done.
BUNDLE  [dev] src/typescript.tsx  100.0% (16/16), done.
BUNDLE  [dev] src/typescript.tsx  100.0% (1/1), done.
```

```
const FileStore = require('metro-cache').FileStore;
const AutoCleanFileStore = require('metro-cache').AutoCleanFileStore;
const HttpStore = require('metro-cache').HttpStore;

module.exports = {
    resolver : {},
    transformer: {},
    serializer: {},
    server: {},
    cacheStores: [
        new AutoCleanFileStore({
            root: './auto',
            intervalMs: 1000,
            cleanupThresholdMs: 30
        }),
        new FileStore({
            root: './cache'
        }),
        new HttpStore({
            endpoint: 'http://localhost:8888/cache/path',
            family: 4, //6
            timeout: 1000
        }),
    ]
}
```


65c6a034bb61f8ac5ed151a7c7a9f2519ffd5a4c91b67302cc9947b794966d629860cd — metro-demo
EXPLORER
OPEN EDITORS
METRO-DEMO
ff
cache
00
0a
0b
0d / 0c
0e
0f
01
1a
1b
1c
1d
1e
1f
02
2a
2b
65c6a034bb61f8ac5ed151a7c7a...
2c
65c6a034bb61f8ac5ed151a7c7a9f2519ffd5a4c91b67302cc9947b794966d629860cd
cache > 2b > 65c6a034bb61f8ac5ed151a7c7a9f2519ffd5a4c91b67302cc9947b794966d629860cd
1 {"dependencies":[],"output":[{"data":{"code":"__d(function (global, _$$_REQUIRE, _$$_IMPORT_DEFAULT, _$$_IMPORT_ALL, modul

Caching

Metro has a multi-layered cache: you can set up multiple caches to be used by Metro instead of one. This has several advantages, on this page we will explain how the caches work.

- FileStore,
- AutoCleanFileStore
- HttpStore

Metro will first look into the `FileStore` when we retrieve a cache. If it can't find the cache there it will check `HttpStore`, and so on. Finally if there's no cache there it will generate a new cache itself. As soon as the cache has been generated, Metro will go again from top to bottom to store the cache in all stores. This also happens if a cache is found. For example, if Metro finds a cache in the `NetworkStore` it will store it in `FileStore` as well.

https://www.youtube.com/watch?v=VMoRoD4YjcE

https://github.com/re-course/metro-bundler/

Yury Pliashkou
CEO, IdeaSoft

@pliashkou
pliashkou@gmail.com

https://www.facebook.com/groups/react.native.belarus/